# An Extreme Course Project -

## CSCI 3060U/SOFE 3980U – Winter 2015

### Project Teams

You are to form a team of three or four people to design, implement, document and deliver a two part software product. All phases to follow Extreme Programming philosophy as much as it applies - in particular,

- continuously maintained test suites as requirements and quality control
- pair programming of all code
- simplest possible solution to every problem
- continuous redesign and rearchitecting
- automation in testing and integration
- frequent integration and complete releases

Every two weeks (or so, see the schedule below) you will deliver concrete evidence of your team's progress as required by project assignments.

### Project Phases

The project will be done in six phases, each of which will be an assignment. Phases will cover steps in the process of creating a quality software result in the context of an Extreme Programming process model.

Assignments will be on the quality control aspects of requirements, rapid prototyping, design, coding, integration and analysis of the product you are building. Throughout the project, you should keep records of all evidence of your product quality control steps and evolution, in order to make the marketing case that you have a quality result at the end of the course.

Your final products will be tested and evaluated.

### Project Schedule

The course project consists of six assignments, with separate handouts for each one. Assignments are scheduled to be due as follows:

- Phase #1: Front End Requirements
  – Wednesday, Feb. 11, 2015
- Phase #2: Front End Rapid Prototype
  – Monday, Feb. 23, 2015
- Phase #3: Front End Requirements Testing
  – Monday, Mar. 9, 2015
- Phase #4: Back End Rapid Prototype
  – Wednesday, Mar. 18, 2015

- Phase #5: Back End Unit Testing
  – Monday, Mar. 30, 2015
- Phase #6: Integration and Delivery
  – Friday, Apr. 10, 2015

**Assignment Hand-Ins**

- Unless otherwise specified, For all assignments *must* be handed in through Blackboard by 11pm on the due date.

For all submissions indicate clearly your team name and all member names and on every hand in.

**Project Requirements**

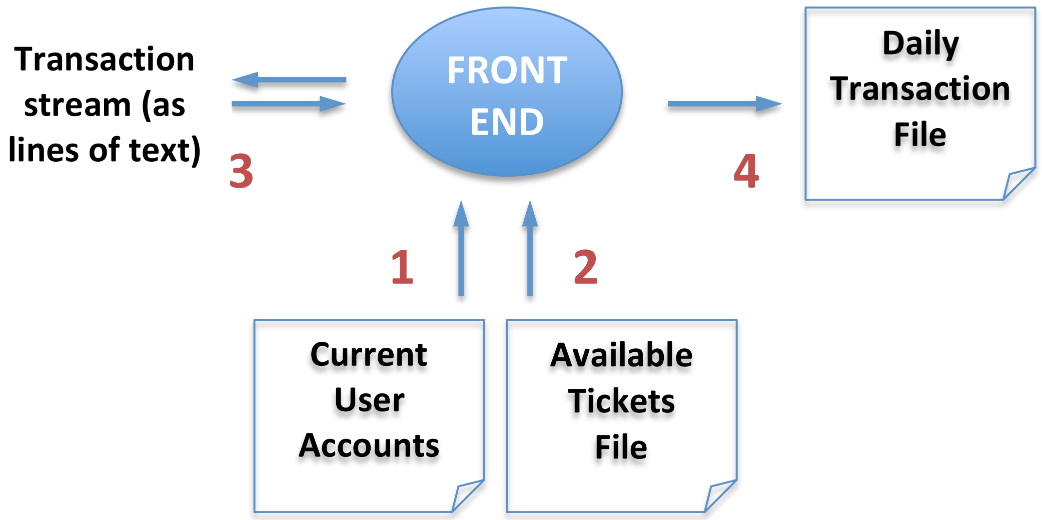
The product you are to design and build is a Ticket Selling Service. The system consists of two parts:

- the Front End, a point of purchase terminal for ticket selling and buying transactions (written in C++)
- the Back End, an overnight batch processor to maintain and update a master ticket file (written in Java)

Both parts will be run as console applications, that is, they are to be invoked from a command line and use text and text file input/output only (this is an important requirement for assignments later in the project, so don't ask for exceptions).

## THE FRONT END

The Front End reads in a file of tickets available for purchase (1) and a file containing information regarding current user accounts in the system (2), it processes a stream of ticket purchase and sale transactions one at a time (3), and it writes out a file of ticket purchase and sale transactions at the end of the session (4).



### Informal Customer Requirements for the Front End

The Front End handles a sequence of transactions, each of which begins with a single transaction code (word of text) on a separate line. The Front End must handle the following transaction codes:

| | |
|---|---|
| login | - start a Front End session |
| logout | - end a Front End session |
| create | - add a user with the ability to buy/sell tickets (privileged transaction) |
| delete | - remove a user (privileged transaction) |
| sell | - sell a ticket or tickets to an event |
| buy | - purchase a ticket or tickets to an event |
| refund | - issue a credit to a buyer's account from a seller's account (privileged transaction) |
| addcredit | - add credit into the system for the purchase of accounts |

## Transaction Code Details

login - start a Front End session

- before processing a login transaction, the Front End reads in the current user accounts file.
- should ask for the username
- after the username is accepted, reads in the available tickets file (see below) and begins accepting transactions
- Constraints:
    - no transaction other than login should be accepted before a login
    - no subsequent login should be accepted after a login, until after a logout
    - after a non-admin login, only unprivileged transactions are accepted
    - after an admin login, all transactions are accepted

logout - end a Front End session

- should write out the daily transaction file (see below) and stop accepting any transactions except login
- Constraints:
    - should only be accepted when logged in
    - no transaction other than login should be accepted after a logout

create – creates a new user with purchasing and/or selling privileges.

- should ask for the new username (as a text line)
- then should ask for the type of user (admin or full-standard, buy-standard, sell-standard)
- should save this information to the daily transaction file
- Constraints:
    - privileged transaction - only accepted when logged in as admin user
    - new user name is limited to at most 15 characters
    - new user names must be different from all other current users
    - maximum credit can be 999,999

delete - cancel any outstanding tickets for purchase or sale and remove the user account.

- should ask for the username (as a text line)
- should save this information for the daily transaction file
- Constraints:
    - privileged transaction - only accepted when logged in as admin user
    - username must be the name of a existing user but not the name of the current user
    - no further transactions should be accepted on a deleted user's available inventory of tickets for sale.

sell – sell a ticket or tickets to an event

- should ask for the event title (as a text line)
- should ask for a sale price for the tickets in dollars (e.g. 15.00)
- should ask for the number of tickets for sale
- should save this information to the daily transaction file
- Constraints:
    - Semi-privileged transaction - only accepted when logged in any type of account except standard-buy.
    - the maximum price for a ticket sale is 999.99
    - the maximum length of an event title is 25 characters
    - the maximum number of tickets for sale is 100
    - no further transactions should be accepted on a new tickets for sale until the next session.

buy – purchase a ticket or tickets to an event

- should ask for the event title, number of tickets and the seller's username
- should display the cost per ticket and the total cost to the user and ask for confirmation in the form of yes or no.
- should subtract the number of tickets from the seller's inventory
- should save this information to the daily transaction file
- Constraints:
    - Semi-privileged transaction - only accepted when logged in any type of account except standard-sell.
    - event title must be a current title with enough tickets available from the specified seller
    - at most 4 tickets can be purchased in one buy transaction (no limit - all remaining tickets - in privileged mode)

refund - issue a credit to a buyer's account from a seller's account (privileged transaction)

- should ask for the buyer's username, the seller's username and the amount of credit to transfer.
- should transfer the specified amount of credit from the seller's credit balance to the buyer's credit balance.
- should save this information for the daily transaction file
- Constraints:
    - Buyer and seller both must be current users

addcredit - add credit into the system for the purchase of accounts

- In admin mode should ask for the amount of credit to add and the username of the account to which credit is being added.
- In a standard account should ask for the amount of credit.
- should save this information to the daily transaction file
- Constraints:
    - In admin mode the username has to be an existing username in the system.
    - A maximum of $1000.00 can be added to an account in a given session.

**General Requirements for the Front End**

The Front End should never crash or stop except as directed by transactions.

The Front End cannot depend on valid input - it must gracefully and politely handle bad input of all kinds (note: but you can assume that input is at least lines of text).

## Daily Transaction File

At the end of each session, when the logout transaction is processed, a daily transaction file for the day is written, listing every transaction made in the session.

Contains variable-length text lines of the form:

```
XX_UUUUUUUUUUUUUUU_TT_CCCCCCCCC
```

Where:

`XX`
: is a two-digit transaction code: 01-create, 02-delete, 06-addcredit, 00-end of session

`UUUUUUUUUUUUUUU`
: is the username (buyer if two users in transaction)

`TT`
: is the user type (AA=admin, FS=full-standard, BS=buy-standard, SS=sell-standard)

`CCCCCCCCC`
: is the available credit.

`_`
: is a space

```
XX_UUUUUUUUUUUUUUU_SSSSSSSSSSSSSSS_CCCCCCCCC
```

Where:

`XX`
: is a two-digit transaction code: 05-refund

`UUUUUUUUUUUUUUU`
: is the buyer's username

`SSSSSSSSSSSSSSS`
: is the seller's username

`CCCCCCCCC`
: is the refund credit.

`_`
: is a space

```
XX_EEEEEEEEEEEEEEEEEEE_SSSSSSSSSSSSS_TTT_PPPPPP
```

where:

`XX`
: is a two-digit transaction code: 03-sell, 04-buy.

`EEEEEEEEEEEEEEEEEEE`
: is the event name

`SSSSSSSSSSSSSS`
: is the seller's username

`TTT`
: Is the number of tickets for sale

`PPPPPP`
: is the price per ticket.

`_`
: is a space

<u>Constraints</u>:

- numeric fields are right justified, filled with zeroes (e.g., 00023 for bank account 23)
- alphabetic fields are left justified, filled with spaces (e.g. John_Doe____________ for bank account holder John Doe)
- unused numeric fields are filled with zeros (e.g., 0000)
- In a numeric field that is used to represent a monetary value, ".00" is appended to the end of the value (e.g. 00110.00 for 110)
- unused alphabetic fields are filled with spaces (blanks) (e.g., ________________________ )
- the sequence of transactions ends with an end of session (00) transaction code

**Current User Accounts File**

Consists of fixed length (28 characters) text lines in the form:

```
UUUUUUUUUUUUUUU_TT_CCCCCCCCC
```

where:

`UUUUUUUUUUUUUUU`
is the username

`TT`
is the user type (AA=admin, FS=full-standard, BS=buy-standard, SS=sell-standard)

`CCCCCCCCC`
is the available credit.

`_`
is a space

Constraints:

- every line is exactly 28 characters (plus newline)
- alphabetic fields are left justified, filled with spaces (e.g., User001________ for username "User001")
- unused numeric fields are filled with zeros (e.g., 0000)
- in a numeric field that is used to represent a monetary value, if the value is only in dollars, then ".00" is appended to the end of the value (e.g. 110.00 for 110)
- unused alphabetic fields are filled with spaces (blanks) (e.g., ________________________ )
- file ends with a special user named END with an empty user type and the credit field empty.

**Available Tickets File**

Consists of fixed length (~~28~~ 45 characters) text lines in the form:

```
EEEEEEEEEEEEEEEEEEE_SSSSSSSSSSSSS_TTT_PPPPPP
```

where:

`EEEEEEEEEEEEEEEEEEE`
is the event name

`SSSSSSSSSSSSS`
is the seller's username

`TTT`
Is the number of tickets for sale

`PPPPPP`
is the price per ticket.

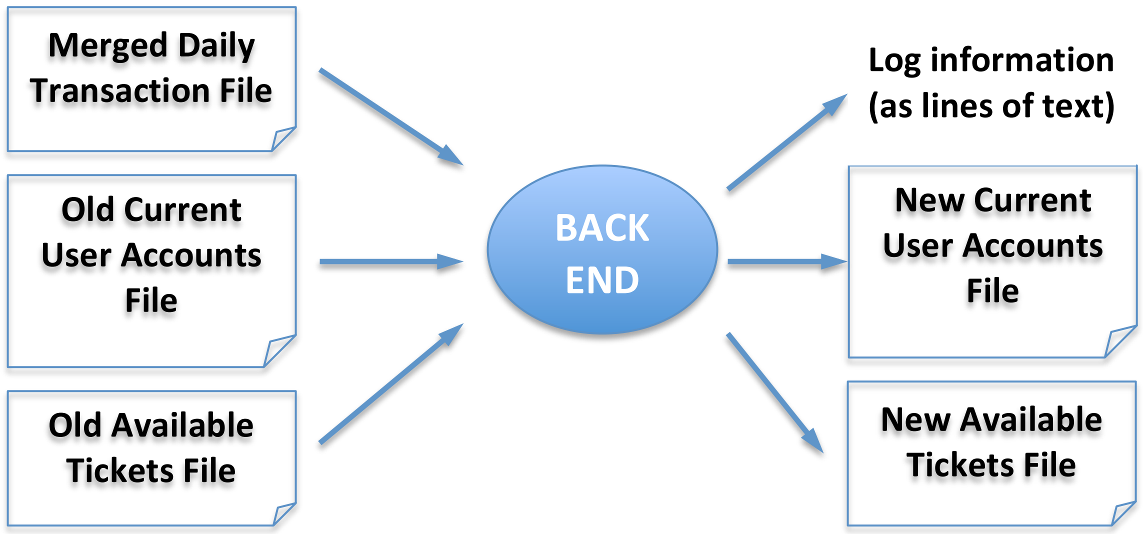
`_`
is a space

Constraints:

- every line is exactly ~~28~~ 45 characters (plus newline)
- alphabetic fields are left justified, filled with spaces (e.g., Celine_Dion_Show________ for event "Celine Dion Show")
- unused numeric fields are filled with zeros (e.g., 0000)
- in a numeric field that is used to represent a monetary value, if the value is only in dollars, then ".00" is appended to the end of the value (e.g. 110.00 for 110)
- unused alphabetic fields are filled with spaces (blanks) (e.g., ________________________ )
- file ends with a special event ticket for sale named END with all other fields empty.

## THE BACK END

The Back End reads in the previous day's User Accounts File and Available Tickets File and then applies all of the daily transactions from a merged set of daily transaction files to these files to produce a new Current User Accounts File and new Available Tickets File for tomorrow's Front End runs.


Merged Daily Transaction File
Old Current User Accounts File
Old Available Tickets File
BACK END
Log information (as lines of text)
New Current User Accounts File
New Available Tickets File

### Informal Customer Requirements for the Back End

The Back End reads the Merged Daily Transaction File (see below) and applies all transactions to the Old Current User Accounts File and Old Available Tickets File to produce the New Current User Accounts File and the New Available Tickets File.

The Back End enforces the following business constraints, and produces a failed constraint log on the terminal as it processes transactions from the merged daily transaction file.

Constraints:

- no event should ever have a negative number of tickets left
- a newly created user must have a title different from all existing users

### General Requirements for the Back End

The Back End should assume correct input format on all files, and need not check for bad input. However, if by chance it notices bad input, it should immediately stop and log a fatal error on the terminal.

## Back End Error Recording

All recorded errors should be of the form: `ERROR: <msg>`

- For failed constraint errors, `<msg>` should contain the type and description of the error and the transaction that caused it to occur.
- For fatal errors, `<msg>` should contain the type and description and the file that caused the error.

## The Merged Daily Transaction File

The Merged Daily Transaction File is the concatenation of any number of Daily transaction files output from Front Ends, ended with an empty one (one containing no real transactions, just a line with a 00 transaction code).